(GAB)
GENERIC ARCNET BOARD MANUAL

Brookhaven National Laboratory
Version 1.0

BY Jack Fried (email jfried@inst.inst.bnl.gov)
Web page (http://www.inst.bnl.gov/~jfried)

INTRODUCTION

The objective behind the Generic ARCNET Board was to create a reliable control system that can be networked and controlled from a single location. The GAB can chain up to 220 nodes per network using a Contemporary Control System HUB. A fiber from the HUB will go to a workstation with an ARCNET interface card to control the network. Multiple ARCNET cards can be placed in the workstation to control many networks of 220 nodes.

The generic ARCNET board was designed for PHENIX Front End Modules (FEM) in mind, however its broad design makes it ideal for any application that requires a reliable slow control system. The software the GAB runs essentially determines its functionality, and with its software download feature it allows the user to change its functionality on the fly.

The GAB was designed to work in a high magnetic field making it capable of running on any PHENIX FEM. The I/O port and the 16 Chip Selects makes its integration into a system straightforward. FEM designers will have to write C code for the GAB, which will be specific to their hardware. A generic program will be made available that can handle standard I/O commands.

HARDWARE:

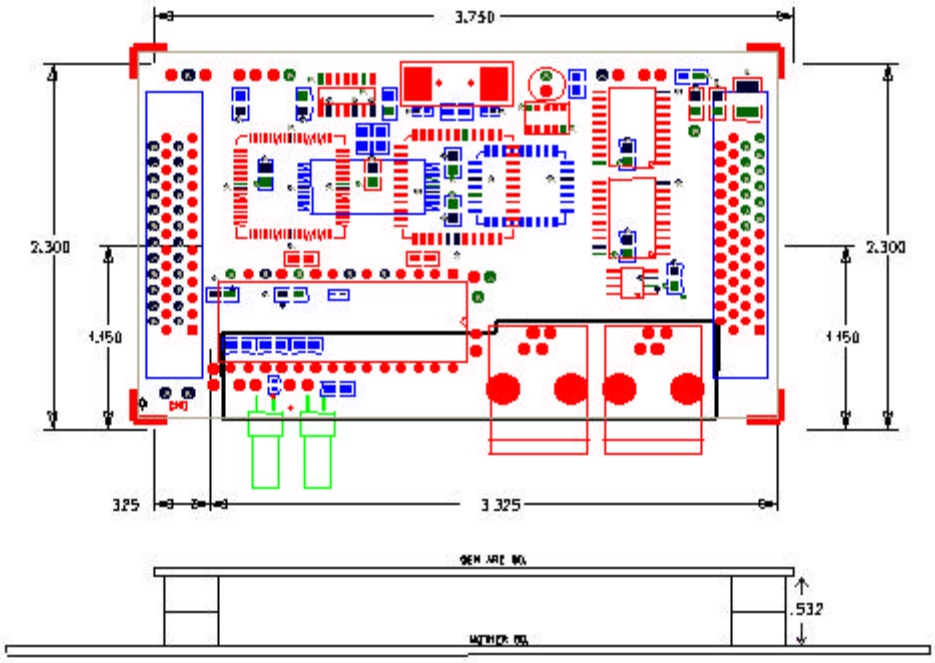
SOFTWARE:

A.

B.

C.

D.



CON2 CON1

Introduction

The Generic ARCNET BOARD (GAB) is a universal microcontroller module based on the SMC COM20051 microcontroller running at a 16 MHZ clock frequency. The microcontroller's integrated ARCNET controller and the on board MAX1480A transceiver provide an optoisolated DC RS-485 ARCNET NODE. The I/O capabilities makes this board especially suitable to serve as an intelligent general purpose node in controlling PHENIX FEM's and other applications.

The GAB can be mounted on an application-specific motherboard via 2 S-BUS connectors (amp #174216-3), through which most of the microcontroller's I/O pins are available.

An 8-bit node ID from S-BUS connectors provide the ARCNET board with an identifier, which in principle enables the use of up to 256 GAB modules (no more than 220 nodes per network is recommended).

The software it runs essentially determines the GAB functionality. The SMC COM20051 can address 64K bytes of RAM and FLASH ROM - 256 bytes from RAM used for memory mapped I/O.

The GAB does not require the use of an emulator; its software can be programmed into the FLASH ROM via ARCNET. This enables rapid system development in allowing the user to test different version of firmware quickly. The firmware can be updated in-system allowing for system upgrades and repairs.

Overview

The GAB features

- 80c31 CPU
- 16 MHZ CLOCK
- 64K RAM
- 64K FLASH ROM
- 8 bit I/O port
- 256 bytes (available for memory mapped I/O)
- 1 serial port OR 2 bit bi-directional I/O port
- 2 bit I/O port (dedicated)
- Automatic reset on power-up
- 16 user definable chip selects
- Two standard 16 bit counters.
- ARCNET software interface built-in
- Firm ware upload feature up to 39K
- Token removal (turn off ARCNET stop noise)
- 2.5Mbits/s data rate (ARCNET)
- works in high magnetic fields (external power supply needed)
- optoisolation between each node.

GAB SIGNALS

GND = 0V VDD = +5V

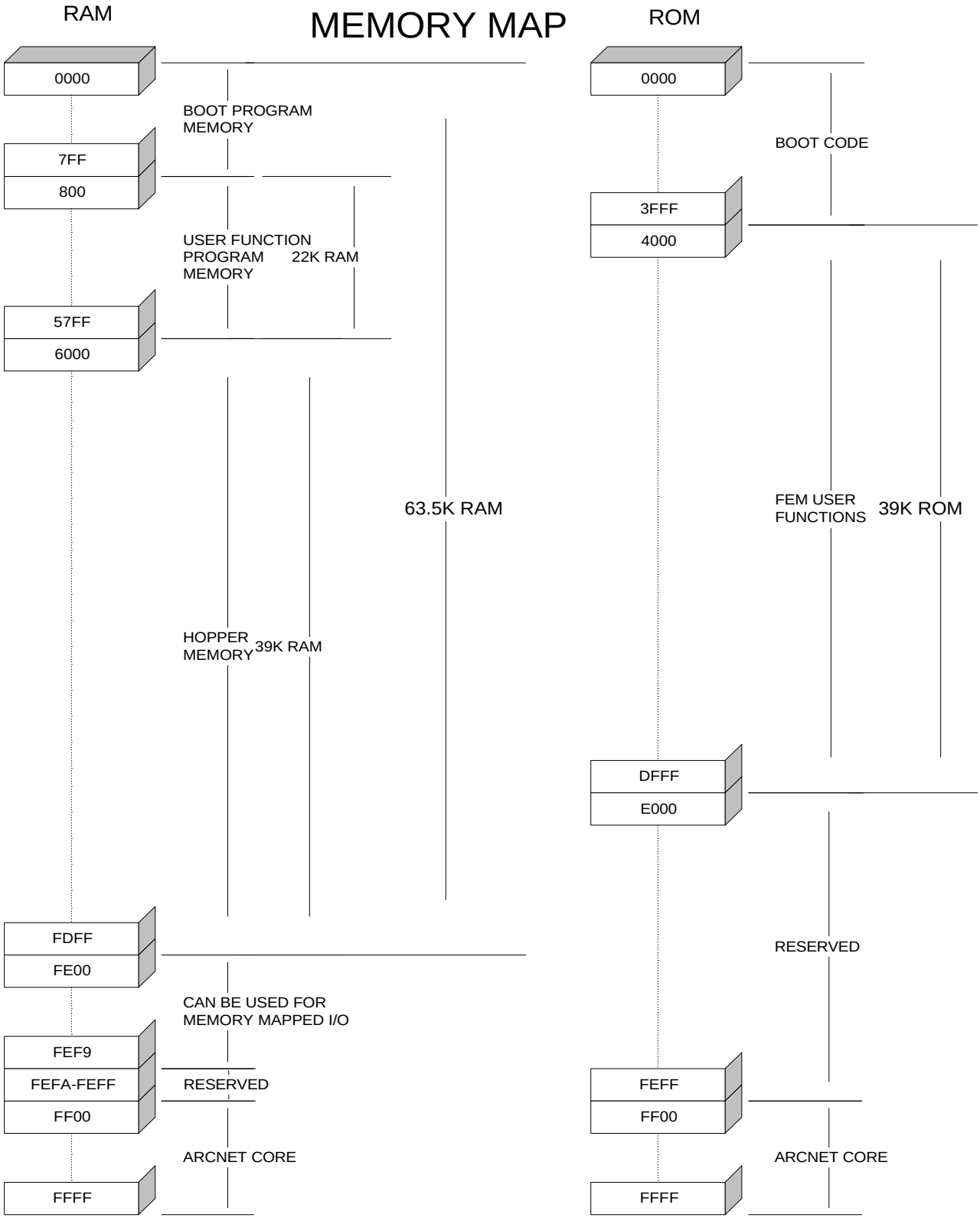
Pin	CON1	CON2
1	CS15	A15O
2	CS13	GND
3	CS11	A13O
4	CS9	GND
5	CS7	A11O
6	CS5	GND
7	CS3	A9O
8	CS1	GND
9	P7O	A7O
10	P5O	GND
11	P3O	A5O
12	P1O	GND
13	PW	A3O
14	PRR	GND

15	VDD	A1O
16	VDD	GND
17	VDD	D7
18	VDD	GND
19	VDD	D5
20	VDD	GND
21	VDD	D3
22	VDD	GND
23	VDD	D1
24	VDD	GND
25	VDD	RD
26	CS14	A14O
27	CS12	GND
28	CS10	A12O
29	CS8	GND
30	CS6	A10O
31	CS4	GND
32	CS2	A8O
33	CS0	GND
34	P6O	A6O
35	P4O	GND
36	P2O	A4O
37	P0O	GND
38	RXD	A2O
39	TXD	GND
40	EXT_RESET	A0O
41	PR	GND
42	IN_RESET	D6
43	ID7	GND
44	ID6	D4
45	ID5	GND
46	ID4	D2
47	ID3	GND
48	ID2	D0
49	ID1	GND
50	ID0	WR

SIGNAL DESCRIPTION.

SIGNAL NAME	DIRECTION	CONNECTOR LOCATION	PIN NUMBER	DESCRIPTION
A0O-A15O	O	CON2	40,15,38,13,36, 11,34,9,32,7,30, 5,28,3,26,1	ADDRESS LINE: Latched address from the SMC COM20051
D0O-D7O	I/O	CON2	48,23,46,21,44, 19,42,17	DATA BUS : Data lines from the SMC COM20051
/RD	O	CON2	25	Active low "read memory "
/WR	O	CON2	50	Active low "write memory"
CS0-CS15	O	CON1	33,8,32,7,31,6,3 0,5,29,4,,28,3,2 7,2, 26,1	CHIP SELECTS: Used to generate levels or pulses and can be used as 2 8-bit output ports. CS0-CS7 located at memory address 0xfefd CS8-CS15 located at memory address 0xfefe
P0O-P7O	I/O	CON1	37,12,36,11,35, 10,34,9	I/O PORT: 8 bit bi-directional port. P1 = "P1.0 – P1.7"
ID0-ID7	I	CON1	50,49,48,47,46, 45,44,43	NODE ID" Used to set the board's unique node id. Located at memory address 0xfeff
/PW	O	CON1	13	PORT WRITE: = P3.3 Active low signal used to indicate the direction of the 8-bit I/O port. When PW is low the I/O port is outputting data.
RXD	I/O	CON1	38	RECEIVE SERIAL: = P3.1 This pin can be used as an single bit I/O or it Can be used with TXD to generate a serial port.
TXD	I/O	CON1	39	TRANSMIT SERIAL: = P3.2 This pin can be used as an single bit I/O or it Can be used with RXD to generate a serial port.
PRR	O	CON1	14	One bit I/O P3.2
PR	I/O	CON1	41	One bit I/O P3.5
EXT_RESET	I	CON1	40	External reset
IN_RESET	O	CON1	42	Internal reset (MAX 708)

MEMORY MAP



GENERIC ARCNET MEMORY SPACE
(COM20051)

GAB POWER REQUIREMENT

The generic ARCNET board power requirements are:

Magnetic field not present (using on board DC to DC converter)

VDD	+5V
I	240ma

Within a magnetic field (power for isolated rs485 supplied externally)

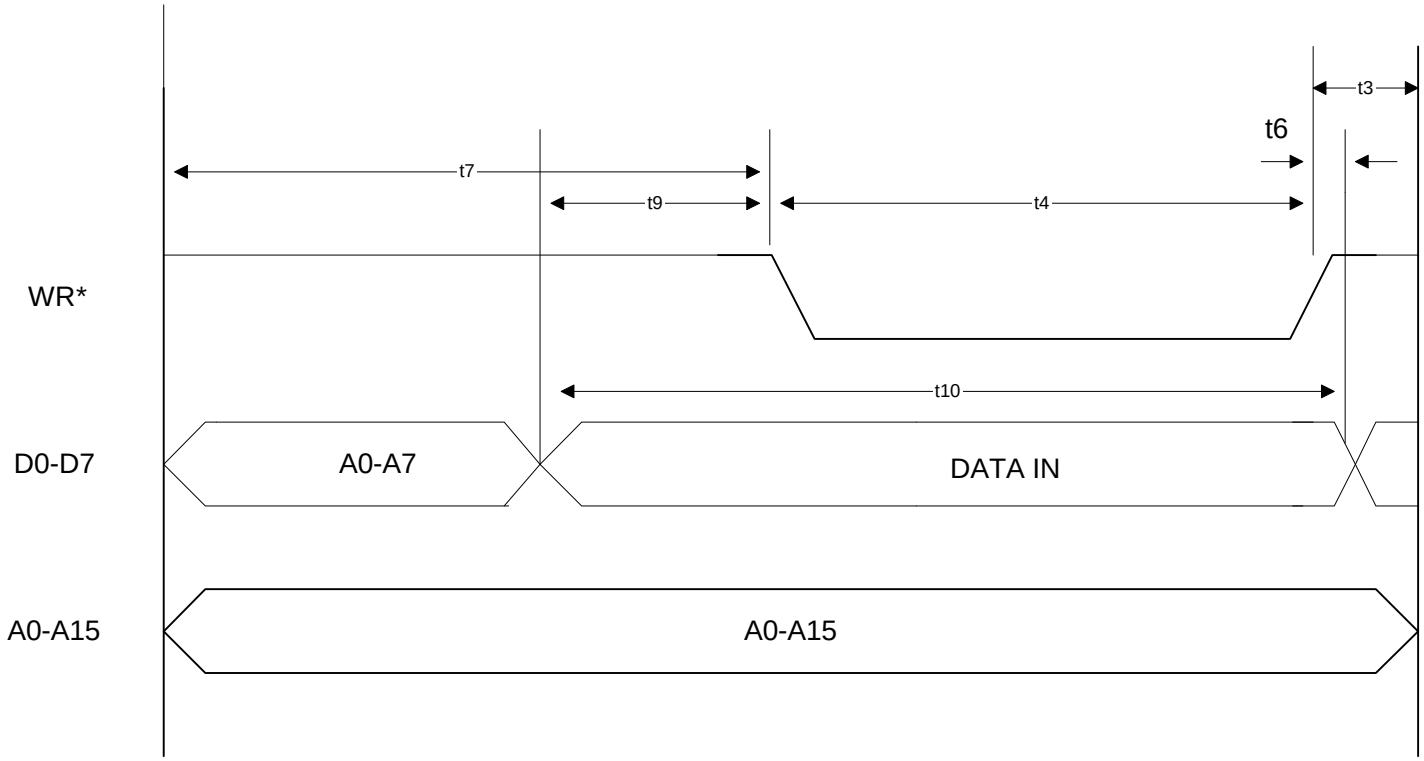
Power from board

VDD	+5V
I	210ma

External power

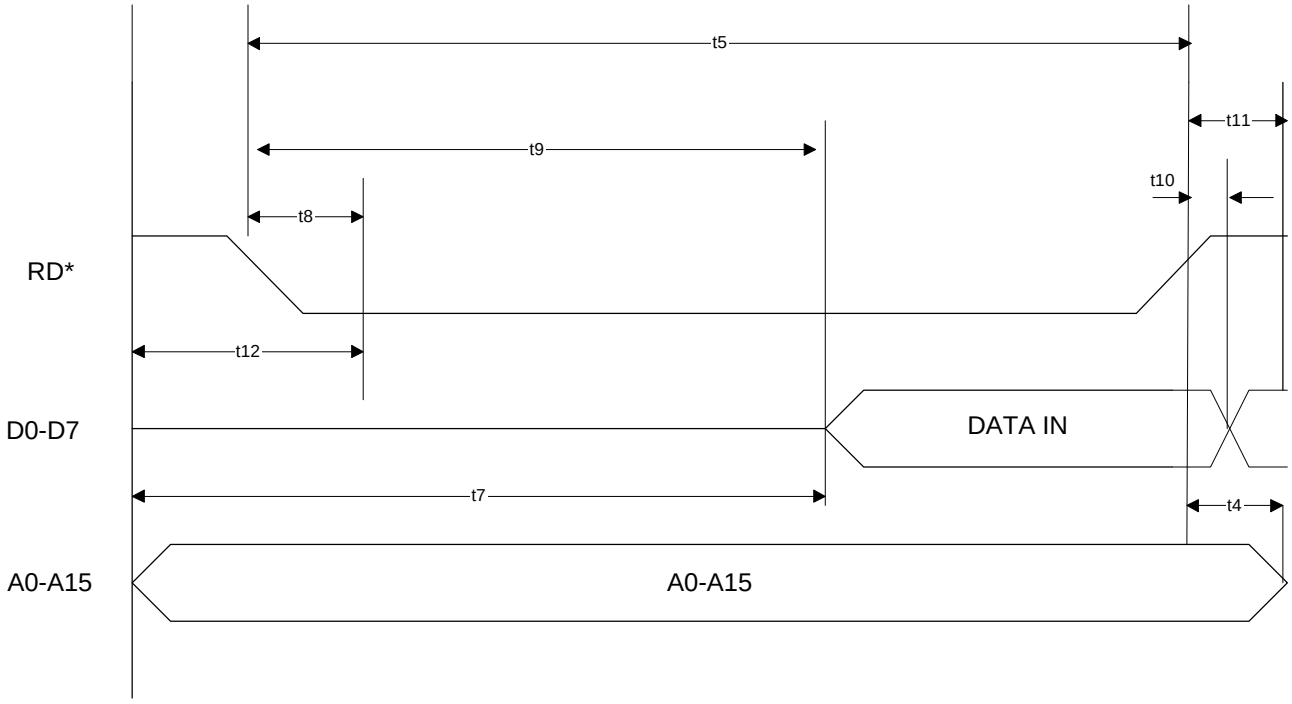
VDD	+5V
I	30ma

EXTERNAL DATA MEMORY WRITE CYCLE



	Parameter	Min	Typ	Max	Units
T3	WR high to Address not valid	22.5			ns
T4	WR Pulse Width	275			ns
T6	Data Hold After WR	12.5			ns
T7	Address Valid to WR low	120			ns
T9	Data Valid to WR Transition	2			ns
T10	Data Valid to WR High	287.5			ns
					ns

EXTERNAL DATA MEMORY READ CYCLE



	Parameter	Min	Typ	Max	Units
T4	RD to Address not valid	22.5			ns
T5	RD Pulse Width	275			ns
T7	Address to Valid in			397.5	ns
T8	RD Low to Address to Float (GAB internal)			0	ns
T9	RD Low to Valid Data In			147.5	ns
T10	Data Hold After RD	0			ns
T11	Data Float After RD			55	ns
T12	Address valid to RD Low	120		0	

SOFTWARE:

INTRODUCTION

The GAB software consists of two parts. The Boot Code (BC) which resides on the protected part of Flash ROM and the FEM User Functions (FUF). The complexity of setting up and controlling the network is part of the BC and is not seen by the FUF thereby simplifying the FUF code. The FUF code is written by the end user and can be uploaded to the GAB at any time.

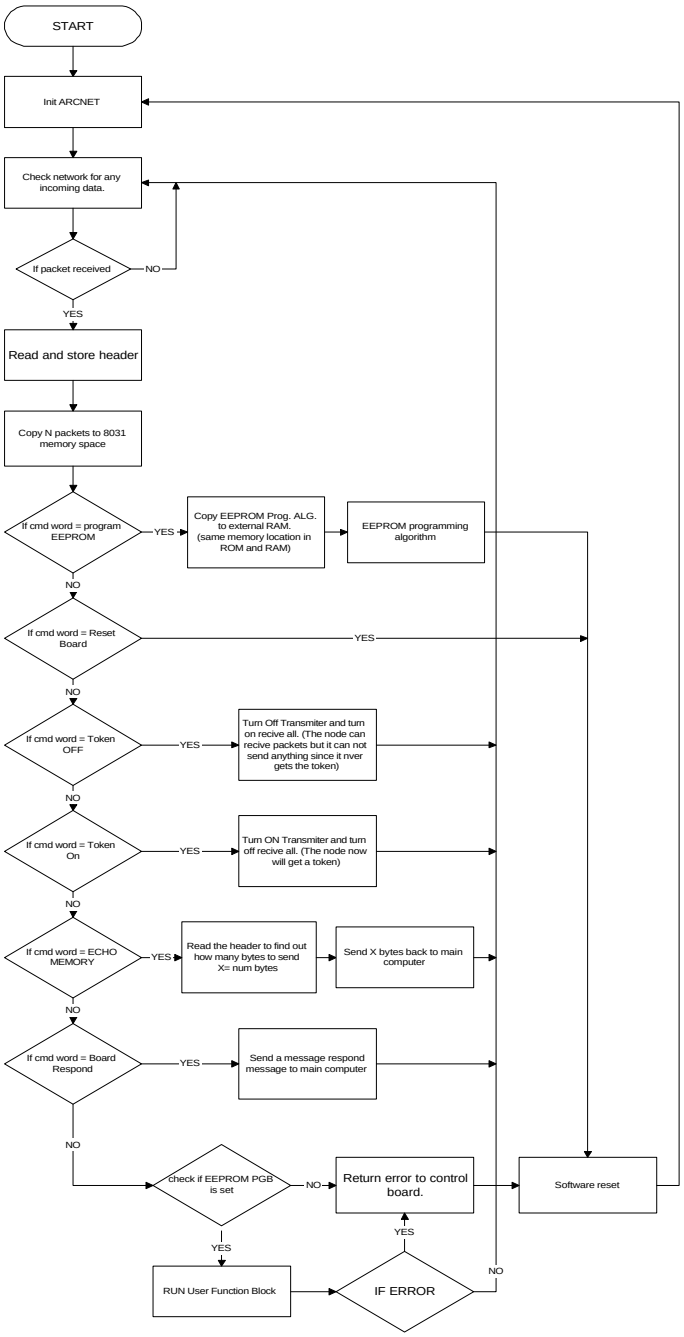
The BC can call either, built in functions or user functions. The BC functions are listed here but are explained in section B.

- PROGRAM FLASH ROM
- RESET BOARD
- TURN OFF TOKEN
- TURN ON TOKEN
- ECHO MEMORY
- BOARD RESPOND
- CALL USER FUNCTION

The network commands that are built into the BC can be accessed from the FUF to send or receive data. The BC commands that are used in the FUF are accessed in the GFEM.H file.

BOOT CODE

The boot code first initializes the ARCNET node and then waits for data to arrive. Once data has arrived and its packet header is analyzed the BC will then ignore or store the packet in HOPPER memory (look at memory map). If there is more than one packet (507 bytes) the BC will continue to receive the rest of the data. After all the data has been received (a maximum of 39 K bytes) the BC will then reread the header and then decide to call the FUF or one of the other BC functions. The boot code flow chart is shown below.



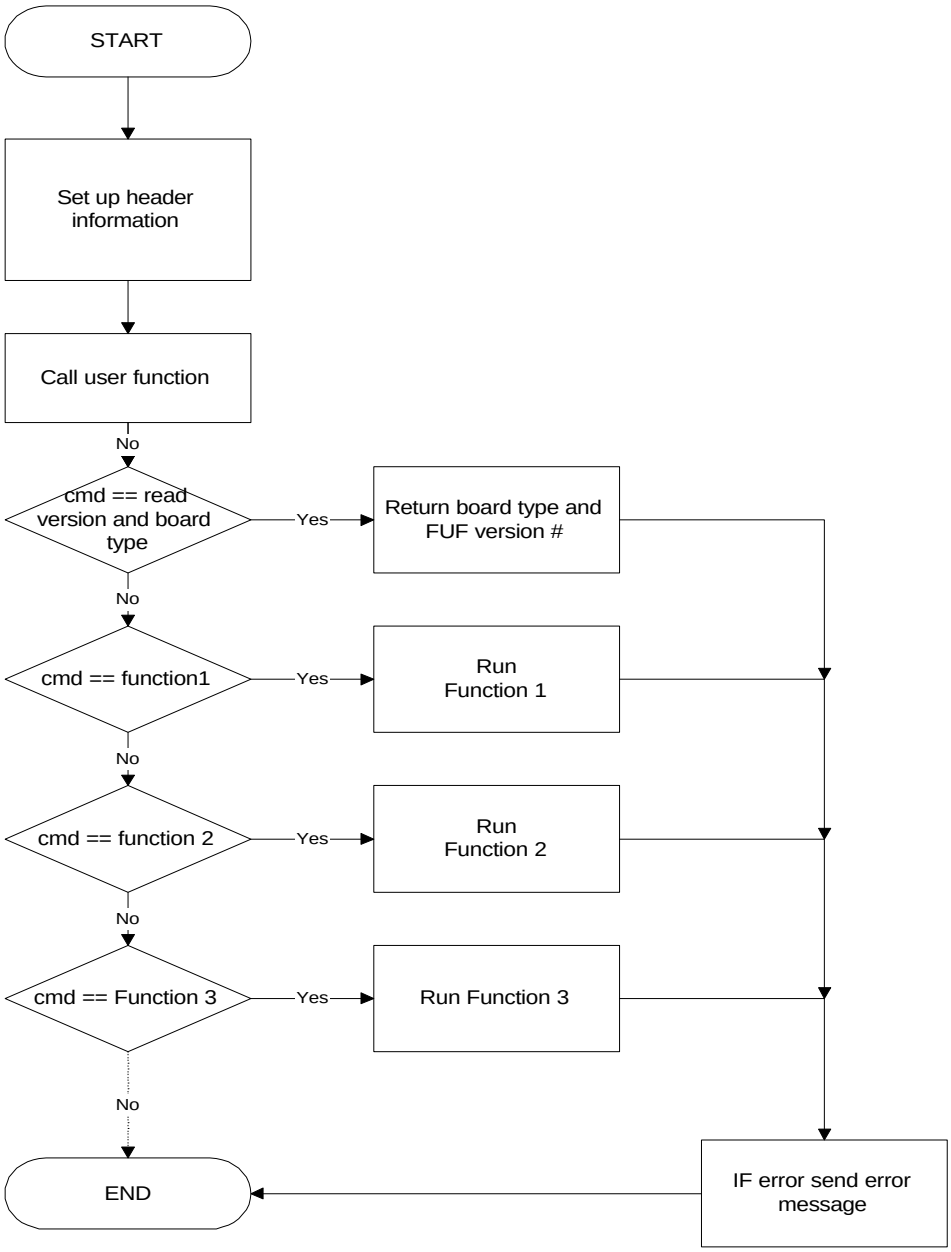
FEM BOOT PROG. ALGORITHM

SOFTWARE REQUIREMENTS

The FUF software requirements are as follows

- The FUF must not run an infinite loop
- The FUF must return to the BC after executing a command
- The FUF must observe the memory boundaries as shown in the memory map
- The FUF software must be linked with cl80.r03 and gfem.h

The BC is not interrupt driven and consequently if the FUF never returns to the BC that node will not respond on the network. It can only be restarted by a hardware reset (a glink reset or a power shut down).



FEM User function block

USER FUNCTION PROGRAM

The user function program structure is laid out in this document. Cretin routines that the FUF will require performing such as receiving and sending data will be discussed in the next few sections. An explanation on how to use the boards I/O devices and chip selects will be also be discussed.

Calling the FUF form the Boot Code

Once the boot code receives a message and stores it in Hopper memory the header is examined. If the command is not recognizes as a standard BC command it calls the FUF. The boot code does this by calling an absolute memory address in ROM where the FUF exists. Once the FUF is called it has full control over the GAB.

The FUF should be treated as a stand-alone program

The FUF should be treated as if it is a stand-alone program. This means that the program has a function called "main" and all libraries such as "stdio.h" need to be include if required (do not expect to use the libraries from the BC to save memory).

The main program should use a case statement. The switch statement will depend on the command being executed.

EXAMPLE

```
#include <stdio.h>
#include <gfem.h>

void  main()
{
 /* Program */
       switch(command)
      {
         case command 1:
         case command 2:
         case command 3:
      }
}
```

Accessing the HOPPER memory.

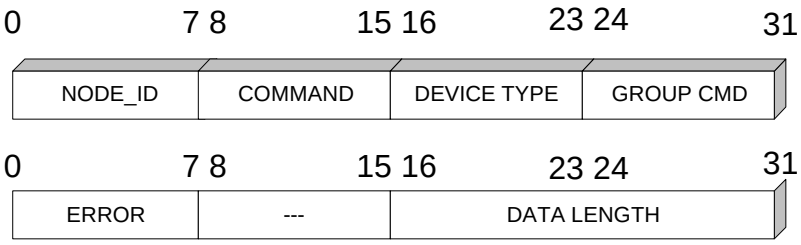
In the header file "gfem.h" a variable called "HOPDATA" is created. This is actually a pointer to the memory location 0x6000h were the data is stored. The HOPDATA variable should be treated as an array of 39KB for example.

EXAMPLE

```
int Read_hopper()
{
  USIGN8 x,y;    /* one byte*/

   x =  HOPDATA[0];
   y = HOPDATA[1];
  return(x+y);
}

void Write_hopper(int x,int y);
{
  int loop;

  HOPDATA[0] = 23;
  HOPDATA[5] = 0x12;
  HOPDATA[7] = (USIGN8) x;    /* x is of type int need to change it to a byte */
  HOPDATA[22] = (USIGN8) y;
  For(loop=0;loop<=20;loop++)
     HOPDATA[loop+100] = 0;
}
```

Accessing the header.

The header like the HOPPER memory is passed along to the FUF as an array. The array HOPCMD contains the information in the follow order.



HOPCMD[0] = node id
HOPCMD[1] = command
HOPCMD[2] =device type
HOPCMD[3] =group command
HOPCMD[4] = ERROR
HOPCMD[5] = xxxxxxx
HOPCMD[6] = (MS) data length
HOPCMD[7] = (LS) data length

A macro in the file “gfem.h” sets up the GAB header into a structure. The macro called “SETUP_PCKT” once called generates a variable called “pckt” the structure is shown below.

```
typedef struct CMD_WORD
{
  USIGN8        cmd;
  USIGN8        node;
  USIGN8        devtype;
  USIGN8        grpcmd;
  unsigned int  dlen;
  USIGN8        error;
}  CMD_WORD;

CMD_WORD   pckt;
```

Example

```
Main()
{
  SETUP_PCKT;
  call_f();
}

void call_f()
{
        switch(pckt.cmd)
        {
            case   0x10:  /*set a device */
                        If(pckt.devtype == DAC)   /*set DAC */
                                Set_DAC();
                        If(pckt.devtype == PRE_AMP)  /* set pre_amp */
                                Set_PEAMP();
                    break;
                                .
                                .
                                .
            case  0x20    /*write port */
                    break;
        }
}
```

How to Send data

Sending data to the main computer requires the use of a function called "SENDBACK(X)" this function is located in the BC and is setup in the file "gfem.h". SENDBACK(x) sends x bytes of data stored in the hopper memory to the main computer.

EXAMPLE

```
 Send_test()
{
  int i;

   for(i=0;i<=100;i++)
     HOPDATA[i] = (USIGN8) i;   /* load hopper memory */
   SENDBACK[100];   /* send 100 bytes to main computer */
   for(i=0;i<=100;i++)
     HOPDATA[100-i] = (USIGN8) i;   /* load hopper memory */
   SENDBACK[100];   /* send another 100 bytes to main computer */
}
```

Calling an ERROR

The ERROR function in the boot code can be accessed by the FUF very simply. As with SENDBACK it is defined in "gfem.h" as "ERROR(x)" where x is the error code to be sent to the main computer. ERROR codes 1-10 are reserved for the BC except for error code '8' which will be used in all FUF programs. The error codes for the BC are listed in section A of this document. All FUFs will have a unique error codes for there specific hardware.

EXAMPLE

```
void call_f()
{
        switch(pckt.cmd)
        {
            case   0x10:   read_mem() /* read memory  */
                 break;
                          .
                          .
                          .
            case  0x20     send_test()  /*write port */
                 break;
default   ERROR(8);  /* error 8 is called when the FUF has no such command in
                        its function list*/
        }
}
```

Using t

NAME	DISCRIPTION	C Name
RXD	I/O BIT or recive serial data	RXD OR P3.0
TXD	I/O BIT or transmit serial data	TXD OR P3.1
PRR	I/O BIT	PRR OR P3.2
PR	I/O BIT	PR OR P3.5
PORT WRITE	PORT 1 t direction Bit	PW or P3.3
PORT 1	8 BIT bi-directional I/O PORT	P1
CHIP SELECT 0- 7	user programable chip selects	LCS
CHIP SELECT 8-15	user programable chip selects	HCS

The table above shows the ports and chip selects that are available to the GAB designer. The next few examples demonstrate how these ports can be used in the FUF program.

SINGLE BIT EXAMPLES

EXAMPLE

```
bit_test()
{
        int x;

        If (PR == 0)  /* check if PR bit is zero */    /* reads the bit*/
                      /* "if ( P3.5==0)" is equivalent to "if(PR==0)"*/
        {
          Printf("the pr bit is zero);
         X =PRR;   /*store  the value in PRR into x;  " 0 or 1" */
        }
}
```

EXAMPLE

```
bit_write()
{

        RXD = 0;        /* set bit RXD to zero*/
```

```
        Read_memory(100);   /*dummy function */
        RXD =1;    /* set bit RXD to one */
        P3.0 = 0;   /* sets RXD to zero again*/
        Printf("rxd is set to zero");
        P3.0 = 1; /* sets it back to one */
}
```

EIGHT BIT PORT EXAMPLES

The 8 bit port uses bit PW or P3.3 to determine the direction of the port. If PW is set to zero the port is set as output port. If set to one it is an input port.

EXAMPLE

```
Port_write()
{
        PW = 0;   /*set port to output */
        P1 = 55;  /* put the value 55 on the port.*/

                /* the port can be accses bit wise as well */
        P1.0 = 0;  /*  set bit 0 to zero */
        P1.1 = 1;  /*  set bit 1  to one */
        P1.2 = 0; /*  set bit 2 to zero */
}
```

EXAMPLE

```
Port_read()
{
        USIGN8   x;    /* create an 8 bit variable */

        PW = 1;  /*set port for input */

        X = P1;  /*store the value of port1 in x */

              /* the port can be accses bit wise as well */

        If(P1.0 == 0)    /* check bit 0 in P1 */
              Printf("the bit is equal to zero in bit 0 of P1");
}
```

EXAMPLE

```
PORT_MIX()
{
        PW =1;  /*set port for read */
        If(P1.0 == 0)    /* check bit 0 in P1 */
        {/
                PW=0; /*set port for write */
                P1  = 00;  /*set p1 to zero */
        }

}
```

CHIP SELECTS EXAMPLES

The CHIP selects can be viewed as two memory mapped 8-bit output ports. Where the first port (CS0-CS7) is located at 0xFEFD and the second (CS8-CS15) at 0xFEFE. Look at the table bellow to see how the chip selects are mapped to the data bits.

Memory Location	D0	D1	D2	D3	D4	D5	D6	D7
0xFEFD	CS0	CS1	CS2	CS3	CS4	CS5	CS6	CS7
0xFEFE	CS8	CS9	CS10	CS11	CS12	CS13	CS14	CS15

EXAMPLE

This example will generate a pulse on a selected chip select line. All Chip Selects in this program are active low.

```
void gen_cs(int x)
{

switch (x) {

        case 0: (* (char xdata *) 0xfefd) = 0xfe;   /* chip select 0*/
                (* (char xdata *) 0xfefd) = 0xff;   /* clear          */
                break;
        case 1: (* (char xdata *) 0xfefd) = 0xfd;   /* chip select 1*/
```

```
        (* (char xdata *) 0xfefd) = 0xff;   /* clear          */
        break;
case 2: (* (char xdata *) 0xfefd) = 0xfb;   /* chip select 2*/
        (* (char xdata *) 0xfefd) = 0xff;   /* clear          */
        break;
case 3: (* (char xdata *) 0xfefd) = 0xf7;   /* chip select 3*/
        (* (char xdata *) 0xfefd) = 0xff;   /* clear          */
        break;
case 4: (* (char xdata *) 0xfefd) = 0xef;   /* chip select 4*/
        (* (char xdata *) 0xfefd) = 0xff;   /* clear          */
        break;
case 5: (* (char xdata *) 0xfefd) = 0xdf;   /* chip select 5*/
        (* (char xdata *) 0xfefd) = 0xff;   /* clear          */
        break;
case 6: (* (char xdata *) 0xfefd) = 0xbf;   /* chip select 6*/
        (* (char xdata *) 0xfefd) = 0xff;   /* clear          */
        break;
case 7: (* (char xdata *) 0xfefd) = 0x7f;   /* chip select 7*/
        (* (char xdata *) 0xfefd) = 0xff;   /* clear          */
        break;

case 8: (* (char xdata *) 0xfefe) = 0xfe;   /* chip select 8*/
        (* (char xdata *) 0xfefe) = 0xff;   /* clear          */
        break;
case 9: (* (char xdata *) 0xfefe) = 0xfd;   /* chip select 9*/
        (* (char xdata *) 0xfefe) = 0xff;   /* clear          */
        break;
case 10: (* (char xdata *) 0xfefe) = 0xfb;   /* chip select 10*/
         (* (char xdata *) 0xfefe) = 0xff;   /* clear          */
         break;
case 11: (* (char xdata *) 0xfefe) = 0xf7;   /* chip select 11*/
         (* (char xdata *) 0xfefe) = 0xff;   /* clear          */
         break;
case 12: (* (char xdata *) 0xfefe) = 0xef;   /* chip select 12*/
         (* (char xdata *) 0xfefe) = 0xff;   /* clear          */
         break;
case 13: (* (char xdata *) 0xfefe) = 0xdf;   /* chip select 13*/
         (* (char xdata *) 0xfefe) = 0xff;   /* clear          */
         break;
case 14: (* (char xdata *) 0xfefe) = 0xbf;   /* chip select 14*/
         (* (char xdata *) 0xfefe) = 0xff;   /* clear          */
         break;
case 15: (* (char xdata *) 0xfefe) = 0x7f;   /* chip select 15*/
         (* (char xdata *) 0xfefe) = 0xff;   /* clear          */
         break;
```

```
   }
}
```

SAMPLE PROGRAM

```
/*
-----------------------------------------------------------
                SAMPLE FUF PROGRAM
-----------------------------------------------------------

*/
#include <gfem.h>
#include <command.h>

                    /* prototype  declarations */

void main(void);
int  call_f();
void setup_pckt();
void fuf_init();
void fuf_ver();
void fuf_wmem(int addr,USIGN8 val);
USIGN8 fuf_rmem(int addr);
void gen_cs_Llev(USIGN8 val);
void gen_cs_Ulev(USIGN8 val);

/*
-------------------------------------------------------------------
                                    MAIN
  void main(void)

          This function is called by the boot program
-------------------------------------------------------------------
*/

void main()
{
  SETUP_PCKT;               /* set up the header in the variable pckt */
  call_f();
}
/* end of main */
```

```
/*
-------------------------------------------------------------------------
                                CALL function
    int call_f()
        This function looks at packet header command and decides which function
        to call. If the function does not exist it will return an error 8 to the
        boot program
-------------------------------------------------------------------------
*/
int  call_f()
{
  int               addr;
 USIGN8         val;
   switch (pckt.cmd)
   {
        case  INIT_BOARD:                   /* init the board */
                                fuf_init();
                                break;
        case  FUF_VER:                      /*sends softwares name and version*/
                                fuf_ver();
                                break;
        case  WRITE_MEMORY:                         /* write a single memory location */
                                addr = ((HOPDATA[0] << 8) + HOPDATA[1]);
                                val = HOPDATA[2];
                                fuf_wmem(addr,val);
                                break;
        case  READ_MEMORY:                          /*read a single memory location */
                                addr = ((HOPDATA[0] << 8) + HOPDATA[1]);
                                HOPDATA[0] = fuf_rmem(addr);
                                SENDBACK(1);  /* send the data to the main computer */
                                break;
        case  READ_PORT:                    /* read the 8 bit I/O port */
                                P3.3 = 1;
                                HOPDATA[0] = P1;
                                SENDBACK(1);  /* send the data to the main  computer */
                                break;
        case  WRITE_PORT:                   /* write the 8 bit I/O port */
                                P3.3 = 0;
                                P1 = HOPDATA[0];
                                break;

        case GEN_CS_LLEV:                   * set the lower chip selects (cs0-cs7)*/
                                            /* to any value */
                                gen_cs_llev(HOPDATA[0]);
```

```
                                    break;
          case GEN_CS_ULEV:                 /* set the upper chip selects (cs8-cs15)*/
                                            /* to any value */
                                    gen_cs_Ulev(HOPDATA[0]);
                                    break;

       default :
                ERROR(8);           /* call ERROR FUNCTION USING function DEFINED IN
GFEM.H  */
                            /* ERROR 8 = NO such CMD*/
                            /* IT SENDS THE ERROR CODE TO THE CONTROL NODE
*/
                return(0);
    }
    return(1);
}
/*
---------------------------------------------------------------------------
                                    FUF INIT
     void fuf_init()
          This function initialized the board

---------------------------------------------------------------------------
*/
void fuf_init()
{
PW = 0;
(* (char xdata *) 0xfefe) = 0xff;   /* clear  cs  */
(* (char xdata *) 0xfefd) = 0xff;   /* clear  cs  */
}

/*
---------------------------------------------------------------------------
                                    FUF version
     void fuf_ver()
          This function sends to the main computer its program name and
          version number

---------------------------------------------------------------------------
*/

void fuf_ver()
{
          HOPDATA[0] =  'g';
          HOPDATA[1] =  'f';
```

```
        HOPDATA[2] =  'u;
        HOPDATA[3] =  'f';
        HOPDATA[4] =  ' ';
        HOPDATA[5] =  'v';
        HOPDATA[6] =  'e';
        HOPDATA[7] =  'r';
        HOPDATA[8] =  ' ';
        HOPDATA[9] =  '1';
        HOPDATA[10] = '.';
        HOPDATA[11] = '0';
         SENDBACK(12);
}

/*
---------------------------------------------------------------------
                                FUF write memory
    void fuf_wmem()
        This function writes a 8 bit value in to the memory location addr

---------------------------------------------------------------------
*/

void fuf_wmem(int addr,USIGN8 val)
{
     (* (char xdata *) addr) = val;
}

/*
---------------------------------------------------------------------
                                FUF read memory
    USIGN8 fuf_rmem()
        This function reads an 8 bit value from the memory location addr

---------------------------------------------------------------------
*/

USIGN8 fuf_rmem(int addr)
{
        return( (* (char xdata *) addr));
}
/*
---------------------------------------------------------------------
                            generate  lower chip select level
    void gen_cs_Llev()
        This function sets the lower (cs0- cs7) chip selects to value determined
```

```
        by val

-------------------------------------------------------------------------
*/
void gen_cs_Llev(USIGN8 val)
{
     LCS = val;   /* set cs */
}



/*
-------------------------------------------------------------------------
                            generate upper  chip select level
    void gen_cs_Ulev()
        This function sets the upper (cs8- cs15) chip selects to value determined
        by val

-------------------------------------------------------------------------
*/


void gen_cs_Ulev(USIGN8 val)
{
     HCS = val;   /* set cs */
}
```

COMMAND.H

```
/* commands  */
#define         WRITE_MEMORY    0x12
#define                 READ_MEMORY     0X13
#define         READ_PORT       0x14
#define         WRITE_PORT      0x15
#define                 GEN_CS_LLEV     0x17
#define         GEN_CS_ULEV     0x18
#define         BIT_PORT_READ   0x19
#define         BIT_PORT_WRITE  0x1a
```

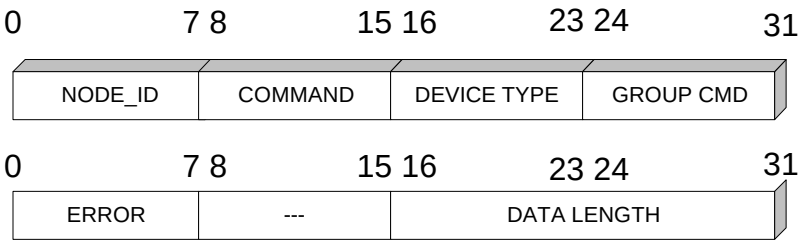
GFEM.H

```
/* ------------------------------------------------------------------------*/
/* Project:      FEM FUNCTIONS                                             */
/* Filename:     GFEM.H                                                    */
/* Description:  set up memory and definitions for GFEM.C                  */
/* Version:      1.1                                                       */
/* Author:       JF                                                        */
/* History:      Original 12/1/97                                          */
/*                                                                         */
/* ------------------------------------------------------------------------*/

#include <io51.h>
#define USIGN8     char

                                        /* data buffers located at 0x6000 to
                                        acommodate the incomming packets */
#define HOPDATA ((USIGN8 *)0x016000)
                                        /* This is the array to accsess the
                                        packet header 8 bytes */
#define HOPCMD  ((USIGN8 *)0x015ff8)
#define GUF     ((int *)0x0107f0)
                                        /* define pointers  for the functions to call
                                        an absolute memory locations */
void (*fblock)(int);
                                /* these are the fixed memory locations
                                        DO NOT CHANGE!!!!  */

#define SENDBACK(x)     {fblock = (void *) GUF[0]; (*fblock) (x);}
#define ERROR(x)        {fblock = (void *) GUF[1]; (*fblock) (x);}

#define RXD   P3.0
#define TXD   P3.1
#define PRR   P3.2
#define PR    P3.5
#define PW    P3.3
#define LCS    (* (char xdata *) 0xfefd)
#define HCS   (* (char xdata *) 0xfefe)
/*
-------------------------------------------------------------------------
    SETUP_PCKT;
        Copy the packet header information  into the pckt structure.
-------------------------------------------------------------------------
*/
```

```
#define SETUP_PCKT  { pckt.cmd = HOPCMD[CMD]; pckt.node =
HOPCMD[NODEID];pckt.devtype=HOPCMD[DEVTYPE];pckt.grpcmd=HOPCMD[G
RPCMD];pckt.dlen= ((HOPCMD[DLEN] << 8) + HOPCMD[1+DLEN]);pckt.error=
HOPCMD[ERR];}


                                    /* packet header information stored in
                                    this structure*/

typedef struct CMD_WORD
{
  USIGN8        cmd;
  USIGN8        node;
  USIGN8        devtype;
  USIGN8        grpcmd;
  unsigned int  dlen;
  USIGN8        error;
}  CMD_WORD;

CMD_WORD   pckt;



#define       PUB           1     /* define  public broadcast */


                /*  header  locations*/
#define       CMD                   1
#define       NODEID                0
#define       DEVTYPE               2
#define       GRPCMD                3
#define       DLEN                  6
#define         UDLEN               6
#define         lDLEN               7
#define                 ERR                 4
#define                 OK                  0



/*        COMMANDS    CODES       */

#define         INIT_BOARD                0x10
#define         FUF_VER           0x11
```

SECTION (A)

GAB HEADER



COMMAND WORD FIG 1.

FIELD	BITS	FUNCTION
NODE_ID	0..7	FEM specific field used to select a node at the front end.
COMMAND	8..15	FEM specific field specifies function for FEM's to perform.
DEVICE TYPE	16..23	Used to select unique device type.
GROUP CMD	24..31	Used to select a group of FEM's
ERROR	0..7	FEM specific field used to return error codes to the control.
----	8..15	NOT DEFINED
DATA LENGTH	16..31	Specifies the total number of bytes in hopper command, including command word and data,

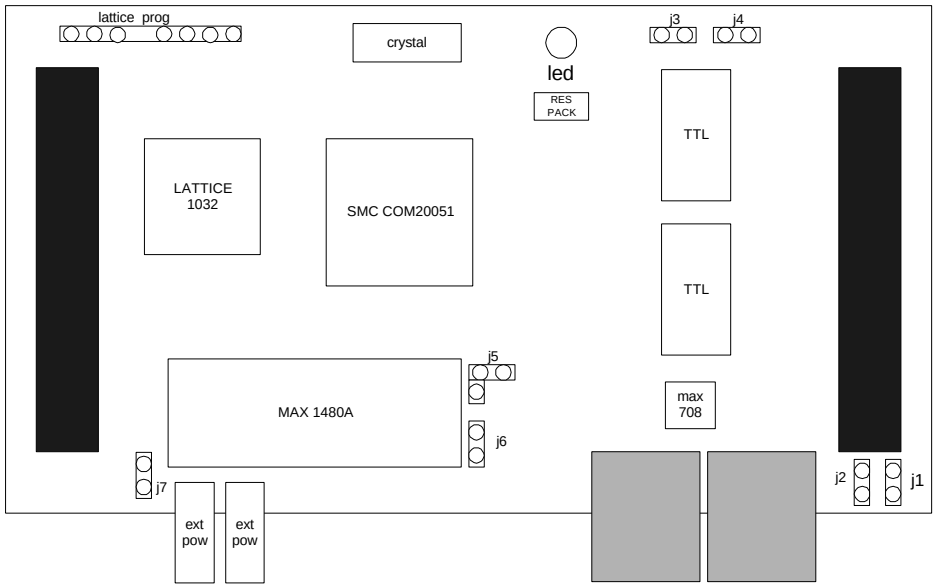
COMMAND WORD TABLE 1.

- The control node has to have a NODE_ID = 0x01.
- The NODE_ ID's = 0x02 – 0x0f and 0xF0 -- 0xFF are reserved.
- The COMMAND field must be placed at the beginning of the hopper memory.
- The GROUP COMMAND = 0x01 is reserved for a broadcast.
- The GROUP COMMAND = 0x00 is reserved for a single node transfer.
- The NODE_ID = 0x00 when sending a packet to more then one node
- The ERROR field must be set to 0x00 when the control node transmits data.
- The ERROR filed will return a value other than zero when an error occurs at a FEM. (data length = 0).
- The ERROR filed will be set to 0x01 when a node is reset for example NODE_ID = 5 & ERROR = 1 ➔ node 5 was reset.
- The control node must be able to receive an ERROR PACKET from any FEM at any given point in time.

GAB ERROR CODES

ERROR CODE	DESCRIPTION
0	**E_OK**
1	**System Reset Occurred**
2	**Bad Packet Received Over Network**
3	**Time Out ERROR**
4	**Flash EEPROM Was Not Programmed Correctly**
5	**PGB NOT SET (FUF section of ROM is empty)**
6	**Incorrect Number of Bytes Received**
7	**Duplicate Node Id detected**
8	**UNKOWN COMMAND CODE**

GAB JUMPER SETTINGS



J1 & J2 biasing resistors

Biasing ON	J1,J2 closed
Biasing OFF	J1,J2 open

J3 & J4 normal or (emulator)

Normal run	J3 closed J4 closed
Emulator	J3 open J4 closed

J5, J6 MAX 1480 isolated power

J7

Internal DC to DC	J5 closed J6,J7 open
External Power (magnetic field)	J5 open J6,J7 closed

SECTION (B)

BOOT CODE FUNCTION CALLS

This section will describe how to call Boot Code functions and describe what they do and what they return. All of the functions use the GAB header shown in Section (A).

PROGRAM FLASH ROM

This function once called can store any data in the hopper memory to the Flash ROM starting at location 0x4000H. This function is typically used to store the user program FUF in the ROM.

Loading the header

Node ID = Node-ID of GAB or '0' for broadcast;
Command = 1
Device Type = don't care
Group_CMD = 0 for single node , 1 for broadcast , 0 for group transfer
ERROR = 0
DATA LEN = xxxx number of bytes to be stored in the ROM

RETURN

One byte set at 77 with command set at 01

ERROR

O = ROM Programming successful
4 = ROM Programming failed

RESET BOARD

This function generates a software reset for the GAB.

Loading

Node ID = Node-ID of GAB or '0' for broadcast;
Command = 2
Device Type = don't care
Group_CMD = 0 for single node , 1 for broadcast , 0 for group transfer
ERROR = 0
DATA LEN = 0

RETURN

NO DATA

ERROR

1 = BOARD RESET

BOARD RESPOND

This function once called echoes back its command with the boards node id.

Loading the header

Node ID = Node-ID of GAB or '0' for broadcast;
Command = 3
Device Type = don't care
Group_CMD = 0 for single node , 1 for broadcast , 0 for group transfer
ERROR = 0
DATA LEN = 0

RETURN

GAB Header Node ID is set to the responding boards ID
NO DATA

ERROR

0 = successful

ECHO MEMORY

This function echo's the contents of the GABS Hopper memory. The amount of data to be echoed is stored in the DATA_LEN parameter of the HEADER.

Loading the header

Node ID = Node-ID of GAB or '0' for broadcast;
Command = 4
Device Type = don't care
Group_CMD = 0 for single node , 1 for broadcast , 0 for group transfer
ERROR = 0
DATA LEN = xxxx Amount of data that will be echoed

RETURN

The contents of the Hopper memory requested

ERROR

0 = successful

TURN OFF TOKEN

This function puts the node in a standby mode where it does not receive the token. When all the nodes of the network are put into this mode the token stops thereby reducing any noise this board would create by the token.

Loading the header

Node ID = '0' for broadcast;
Command = 5

Device Type = don't care
Group_CMD = 1 for broadcast
ERROR = 0
DATA LEN = 0

RETURN

NO DATA (NODE OFF LINE)

TURN ON TOKEN

This function takes the node off a standby mode so that it does receive the token.

Loading the header

Node ID = '0' for broadcast;
Command = 6
Device Type = don't care
Group_CMD = 1 for broadcast
ERROR = 0
DATA LEN = 0

RETURN

NO DATA (NODE ON LINE)

ACTIVE NETWORK MAP

This function once called echoes back its command with the boards node id.

Loading the header

Node ID = Node-ID of GAB or '0' for broadcast;
Command = 7
Device Type = don't care
Group_CMD = 0 for single node , 1 for broadcast , 0 for group transfer
ERROR = 0
DATA LEN = 0

RETURN

GAB Header Node ID is set to the responding boards ID

NO DATA

ERROR

0 = successful

BOOT CODE VERSION NUMBER

This function once called sends back its version number in the data field.

Loading the header

Node ID = Node-ID of GAB or '0' for broadcast;
Command = 8

Device Type = don't care
Group_CMD = 0 for single node , 1 for broadcast , 0 for group transfer
ERROR = 0
DATA LEN = 0

RETURN

GAB Header Node ID is set to the responding boards ID
The boot code version number.

ERROR

0 = successful

CALL USER FUNCTION

This function calls the FUF

Loading the header

Node ID = Node-ID of GAB or '0' for broadcast;
Command = (0x10 - 0xFF)
Device Type = user defined
Group_CMD = 0 for single node , 1 for broadcast , 0 for group transfer
ERROR = 0
DATA LEN = xxxx number of bytes to be transfered

RETURN

User defined

ERROR

5 = No program in ROM
8 = No command is not a user function
x = USER DEFINED

SECTION (C)

What you need to get started.

1. ARCNET interface board for the PC.

   Can be purchased for Contemporary Control System, INC.
   Part number : PCX20-485
   Tel number : 708 963-7070
   Web page : http://www.ccontrol.com

2. IAR 8051 C compiler

   PHENIX groups can

   CALL Jack Fried at (516) 344-4441
   Or send email to: jfried@inst.inst.bnl.gov

   ALL other groups must purchase the compiler form:

   IAR SYSTEMS SOFTWARE, INC
   1 Maritime Plaza
   San Francisco, CA 94111
   (415) 765-5500

3. A GENERIC ARCNET BOARD (GAB)

   CALL Jack Fried at (516) 344-4441
   Or send email to: jfried@inst.inst.bnl.gov

4. Software tool kit for the GAB

   Can be Downloaded form the web at
   http://www.inst.bnl.gov/~jfried/arcnet

5. This manual